Ministry of Education of Moldova

Technical University of Moldova

Faculty "Computers, Informatics and Microelectronics"

Report

Laboratory work nr. 2
on Embedded systems

Performed by:
st. gr. FAF-141 D. Lupei

Verified by:
assoc. prof., A. Bragarenco

Chisinau -2016

Topic

General Purpose Input/Output registers on AVR. LED and interfacing LCD.

Objectives

-Understanding GPIO

-

Tasks

Write a C program and schematics for Micro Controller Unit (MCU) using led which will be turned on by pushing on button and turned off when button is released. Additionally use LCD Display which will display current state of led.

Short theory

General-purpose input/output (GPIO) is a generic pin on an integrated circuit or computer board whose behavior—including whether it is an input or output pin—is controllable by the user at run time. GPIO, pins are the digital I/O for the AVR family. These pins are true push-pull outputs. The AVR can drive a high or low level, or configure the pin as an input with or without a pull-up. GPIOs are grouped into "ports" of up to 8 pins, though some AVRs do not have enough pins to provide all 8 pins in a particular port, e.g. the Mega48/88/168 does not have a PortC7 pin. Control registers are provided for setting the data direction, output value (or pull-up enabled), and for reading the value on the pin itself. An individual pin can be accessed using bitwise manipulation instruction

Each port has 3 control registers associated with it, DDRx, PORTx, and PINx. Each bit in those registers controls one GPIO pin, i.e. bit 0 in DDRA controls the data direction for PortA0 (often abbreviated PA0), and bit 0 in PORTA will control the data (or pullup) for PA0.

- **DDRx** – Data Direction Register

  DDRx configures data direction of port pins. Means its setting determines whether port pins will be used for input or output. Writing 0 to a bit in DDRx makes corresponding port pin as input, while writing 1 to a bit in DDRx makes corresponding port pin as output.

- **PORTx** – Pin Output Register

  PORTx is used for two purposes.

  1) To output data ： when port is configured as output
  2) To activate/deactivate pull up resistors – when port is configures as input

  When you set bits in DDRx to 1, corresponding pins becomes output pins. Now you can write data into respective bits in PORTx register. This will immediately change state of output pins according to data you have written. In other words to output data on to port pins, you have to write it into PORTx register.

- **PINx** – Pin Input Register

  PINx used to read data from port pins. In order to read the data from port pin, first you have to change port's data direction to input. This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give you data that has been output on port pins.

Work Process:

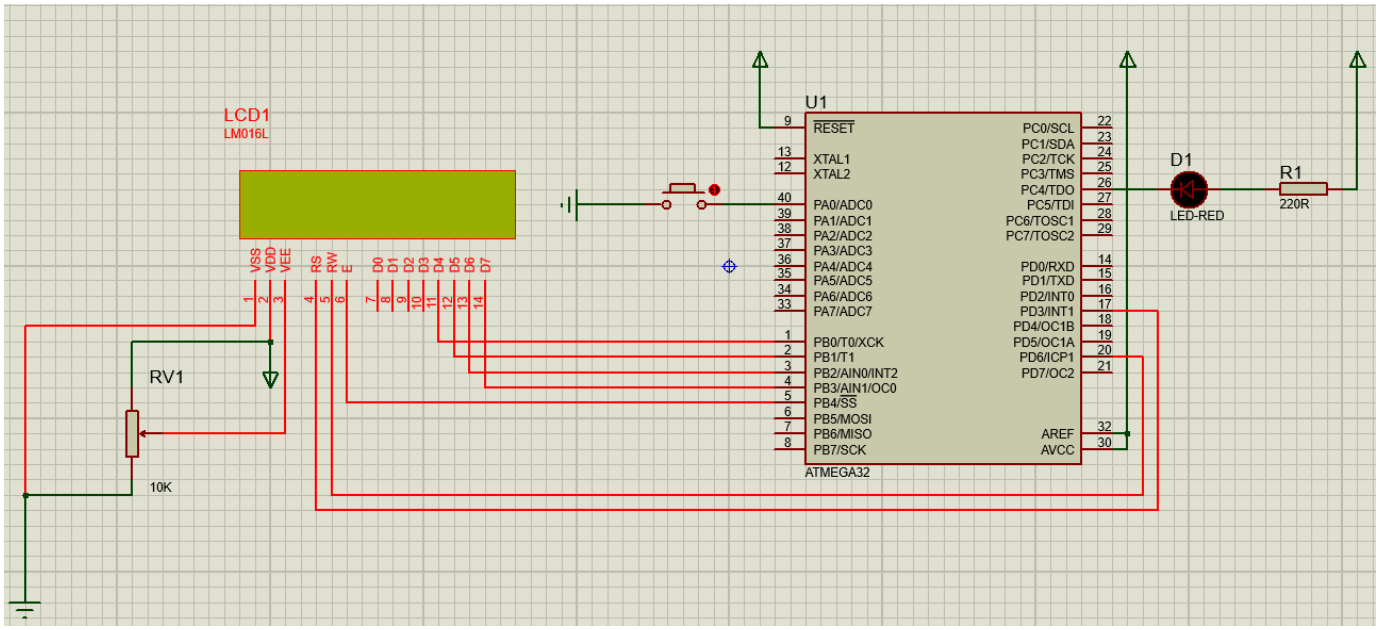
1 .In order to start we must create the circuit scheme with the simulation software Proteus where we need

1. Atmega32 MCU
2. LCD LM016L (16x2)
3. LED
4. Push Button

We use LCD LM016L because The most commonly used LCDs found in the market today are 1 Line, 2 Line or 4 Line LCDs which have only 1 controller and support at most of 80 charachers, whereas LCDs supporting more than 80 characters make use of 2 HD44780 controllers.
Most LCDs with 1 controller has 14 Pins and LCDs with 2 controller has 16 Pins (two pins are extra in both for back-light LED connections). Pin description is shown in the table below.

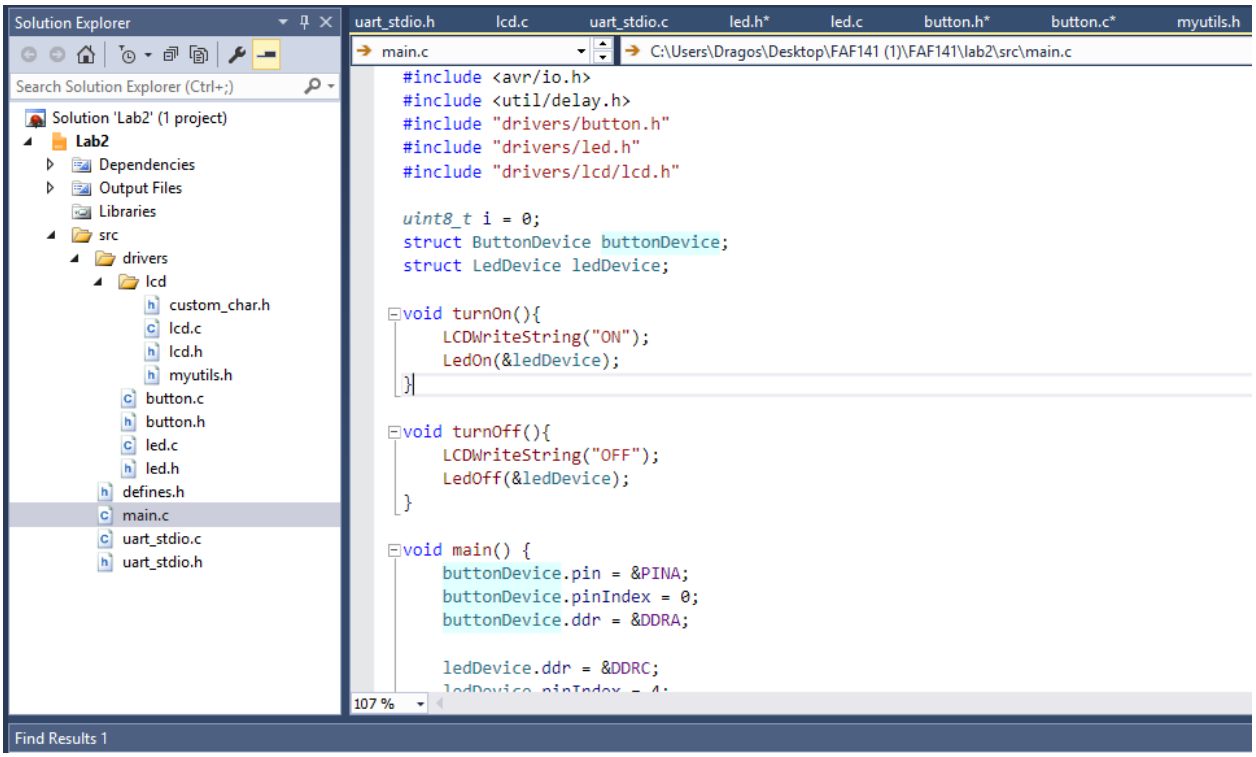
Pin No.	Name	Description
Pin no. 1	**VSS**	Power supply (GND)
Pin no. 2	**VCC**	Power supply (+5V)
Pin no. 3	**VEE**	Contrast adjust
Pin no. 4	**RS**	0 = Instruction input 1 = Data input
Pin no. 5	**R/W**	0 = Write to LCD Module 1 = Read from LCD module
Pin no. 6	**EN**	Enable signal
Pin no. 7	**D0**	Data bus line 0 (LSB)
Pin no. 8	**D1**	Data bus line 1
Pin no. 9	**D2**	Data bus line 2
Pin no. 10	**D3**	Data bus line 3
Pin no. 11	**D4**	Data bus line 4
Pin no. 12	**D5**	Data bus line 5
Pin no. 13	**D6**	Data bus line 6
Pin no. 14	**D7**	Data bus line 7 (MSB)

Table 1: Character LCD pins with 1 Controller



2 .After designing the scheme we have to implement the programf itself and to write code according to the draw flowchart

3. This project structure is the following:



4. I used **avr/delay.h** library for **_delay_ms** procedure used in main program and **avr/io.h** for drivers

To use LCD, LED and Button we should define drivers for each of them

Both LED driver and Button driver have dependencies on #include <avr/io.h>

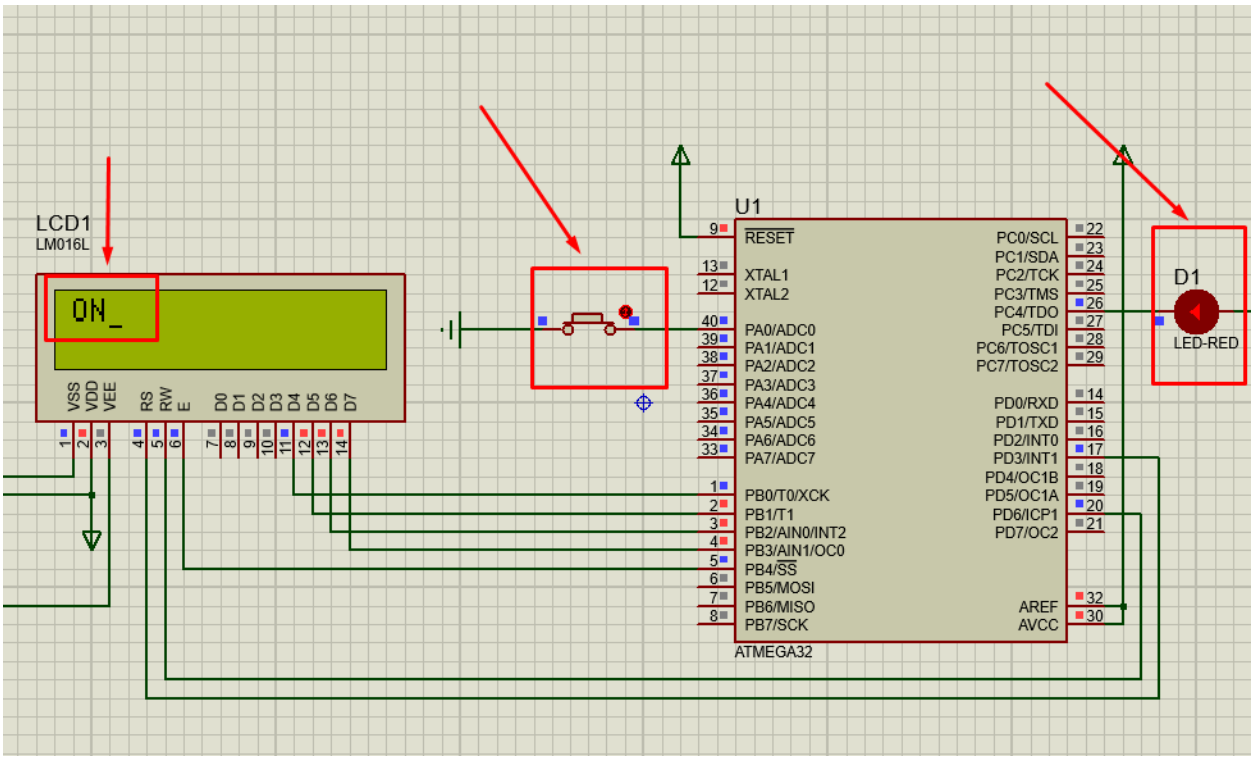
```
struct LedDevice {
	uint8_t pinIndex;
	volatile uint8_t *ddr;
	volatile uint8_t *port;
};

void LedInit(struct LedDevice *device); //Initialization
void LedOn(struct LedDevice *device);// Turn On LED
void LedOff(struct LedDevice *device); // Turn Off LED

struct ButtonDevice {
	uint8_t pinIndex;
	volatile uint8_t *pin;
	volatile uint8_t *ddr;
};
void ButtonInit(struct ButtonDevice *device); //Initailization
char ButtonPressed(struct ButtonDevice *device); //Check Button State
```

The structures contains data for drivers configuration in order to work properly with respective device.

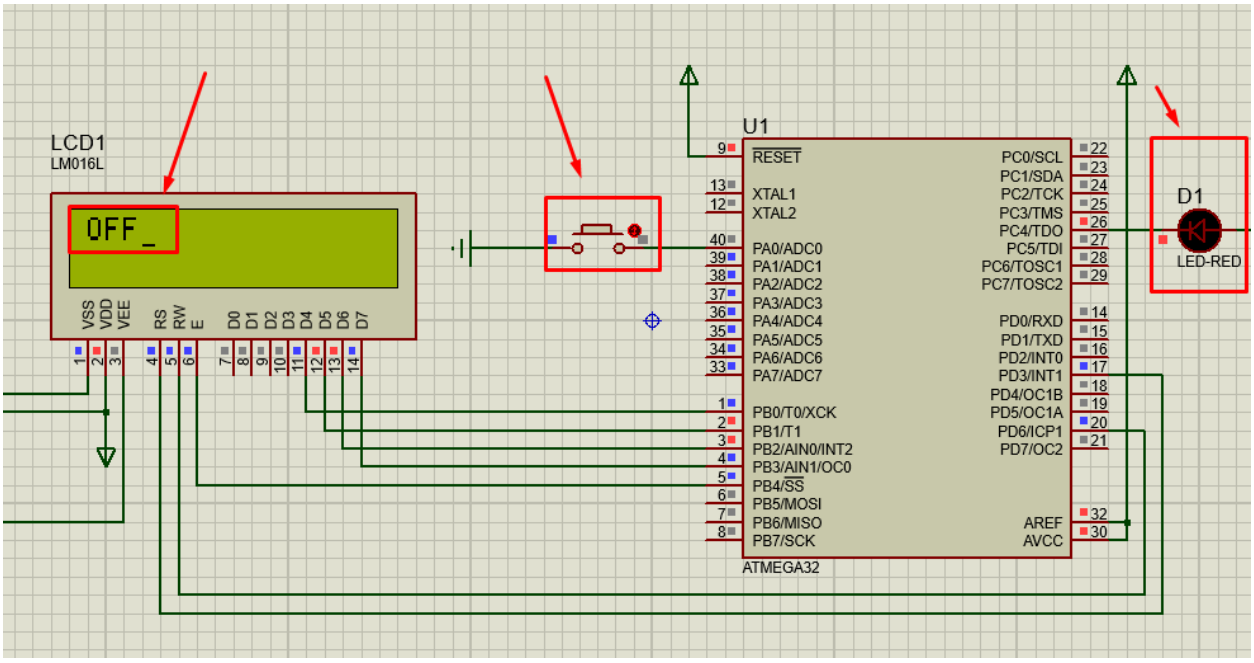
5. The main program defines
    - Defines configurations for Led Device ,
    - Button Device,
    - Initializes LED,
    - Intilializes Button
    - Initiliazes LCD display.
    - Starts infinite loop that check
        a. if button is pressed LED will turn on and on LCD will be displayed ON message
        b. if button is not pressed LED will be off and on LCD will be displayed OFF message
        c. sleep for 100ms

After writing and checking code for error if it was everthing okay we build the solution with resulted HEX and ELF files which we will set in proteus in order to simulate the result.

Simulation when button is pressed



Simulation when button is not pressed



Conclusion:

After doing this laboratory work I learned a little bit more about microcontrollers and how to work with drivers. I get the basic knowledge in understanding programming for standart Input/Output and configuring these devices and more about how to connect a peripheral to microcontroller receive and sent data trough PIN and PORT registers.

Bibliography

- M.A. Mazidi, "*AVR microcontroller and embedded system using C and assembly language*", 2009
- Thomas Bräunl, "*Embedded Robotics*"
- http://extremeelectronics.co.in/avr-tutorials/using-lcd-module-with-avrs/
- https://www.draw.io/
- http://extremeelectronics.co.in./
- http://www.microlab.club/search/label/Laborator
- https://embeddedcenter.wordpress.com/ece-study-centre/display-module/lcd-16x2-lm016l/

Appendix

main.c

```
#include <avr/io.h>
#include <util/delay.h>
#include "drivers/button.h"
#include "drivers/led.h"
#include "drivers/lcd/lcd.h"

uint8_t i = 0;
struct ButtonDevice buttonDevice;
struct LedDevice ledDevice;

void turnOn(){
	LCDWriteString("ON");
	LedOn(&ledDevice);
}

void turnOff(){
	LCDWriteString("OFF");
	LedOff(&ledDevice);
}

void main() {
	buttonDevice.pin = &PINA;
	buttonDevice.pinIndex = 0;
	buttonDevice.ddr = &DDRA;

	ledDevice.ddr = &DDRC;
	ledDevice.pinIndex = 4;
	ledDevice.port = &PORTC;

	ButtonInit(&buttonDevice);
	LedInit(&ledDevice);
	LCDInit(LS_ULINE);

	while(1){
		LCDClear();
		if(ButtonPressed(&buttonDevice)){
			turnOn();
		} else {
			turnOff();
		}
		_delay_ms(100);
	}
}
```

led.c

```
#include "led.h"

void LedInit(struct LedDevice *device){
	*(device->ddr) |= 1 << device->pinIndex;
}

void LedOn(struct LedDevice *device){
	*(device->port) &= ~(1<< device->pinIndex);
}

void LedOff(struct LedDevice *device){
	*(device->port) |= (1<< device->pinIndex);
}
```

led.h

```
#ifndef LED_H_
#define LED_H_
#include <avr/io.h>

struct LedDevice {
	uint8_t pinIndex;
	volatile uint8_t *ddr;
	volatile uint8_t *port;
};

void LedInit(struct LedDevice *device);
void LedOn(struct LedDevice *device);
void LedOff(struct LedDevice *device);

#endif /* LED_H_ */
```

button.c

```
#include "button.h"


void ButtonInit(struct ButtonDevice *device){
	*(device->ddr) = ~(1<<device->pinIndex);
}

char ButtonPressed(struct ButtonDevice *device){
	if(~(*(device->pin))&(1<<device->pinIndex))
		return 1;
	else
		return 0;
}
```

button.h

```
#ifndef BUTTON_H_
#define BUTTON_H_
#include <avr/io.h>

struct ButtonDevice {
	uint8_t pinIndex;
	volatile uint8_t *pin;
	volatile uint8_t *ddr;
};

void ButtonInit(struct ButtonDevice *device);
char ButtonPressed(struct ButtonDevice *device);

#endif /* BUTTON_H_ */
```